Classification Level: Top Secret ( ) Secret ( ) Internal ( ) Public (√)

Rockchip_User_Guide_RKNN_API_V1.3.2_EN

(Technology Department, Graphic Display Platform Center)

Mark: [ ] Editing [√] Released	Version	V1.3.2
	Author	Kevin Du
	Completed Date	2020-04-02
	Reviewer	Randall
	Reviewed Date	2020-04-02

Revision History

Version	Modifier	Date	Modify description	Reviewer
V0.9.1	Kevin Du	2018-11-27	Initial release.	Randall
V0.9.2	Kevin Du	2018-12-19	Mainly modify the API definition of input and output.	Randall
V0.9.3	Kevin Du	2019-01-24	Add API migration instruction from v0.9.1 to v0.9.2.	Randall
V0.9.4	Kevin Du	2019-03-11	Fix the issue that channel_mean is not in effect.	Randall
V0.9.6	Kevin Du	2019-05-14	Add rknn_init2 Function.	Randall
V0.9.7	Kevin Du	2019-06-13	Add x86 linux support.	Randall
V0.9.8	Kevin Du	2019-06-26	1. Update Linux X86 Demo section. 2. Add support of rknn_batch_size > 1. 3. Add query function of the devices ID list.	Randall
V0.9.9	Kevin Du	2019-07-16	1. Add support of multi-input 2. Fix inference error when input channel > 3 3. Modify the name of document	Randall
V1.2.0	Kevin Du	2019-09-17	Unified Version to V1.2.0	Randall
V1.3.0	Kevin Du	2019-11-27	1. Add multi-channels-mean support 2. Unified Version to V1.3.0	Randall
V1.3.2	Kevin Du	2020-04-02	1. Fix issue that API version missing version number. 2. update multi-context support. 3. fix the data convert overflow when dst format is int8/uint8/int16 4. add AVX support for fp16 convert. (x86) 5. speed up load model in multi-thread. 6. update Version to V1.3.2	Randall

Table of Contents

1 OVERVIEW 4

2 SYSTEM DEPENDENCIES DESCRIPTION 4

2.1 LINUX PLATFORM DEPENDENCIES 4

2.2 ANDROID PLATFORM DEPENDENCIES 4

3 API INSTRUCTIONS 4

3.1 RKNN API DETAILS 5

3.1.1 rknn_init & rknn_init2 5

3.1.2 rknn_destroy 7

3.1.3 rknn_query 7

3.1.4 rknn_inputs_set 11

3.1.5 rknn_run 12

3.1.6 rknn_outputs_get 12

3.1.7 rknn_outputs_release 14

3.1.8 rknn_find_devices 15

3.2 RKNN DATA STRUCTURE DEFINITION 15

3.2.1 rknn_input_output_num 15

3.2.2 rknn_tensor_attr 15

3.2.3 rknn_input 17

3.2.4 rknn_output 18

3.2.5 rknn_perf_detail 18

3.2.6 rknn_perf_run 19

3.2.7 rknn_init_extend 19

3.2.8 rknn_run_extend 19

3.2.9 rknn_output_extend 20

3.2.10 rknn_sdk_version.....*20*

3.2.11 rknn_devices_id.....*21*

3.2.12 Error Code.....*21*

3.3 RKNN API BASIC CALL FLOW.....21

4 DEMO INSTRUCTIONS.....28

4.1 LINUX ARM DEMO.....28

4.1.1 Compilation Instructions.....*28*

4.1.2 Run Instructions.....*28*

4.2 LINUX X86 DEMO.....29

4.2.1 Compilation Instructions.....*29*

4.2.2 Run Instructions.....*30*

4.3 ANDROID DEMO.....31

4.3.1 Compilation Instructions.....*31*

4.3.2 Run Instructions.....*32*

5 APPENDIX.....33

5.1 API MIGRATION INSTRUCTIONS.....33

1 Overview

The RKNN API is an NPU(Neural Network Unit) acceleration interface based on Linux/Android. It provides general acceleration support for AI related applications.

This manual mainly consists of three parts.

1）RKNN API: Detailed API definition and instructions for using.
2）Linux Demo: Compile the Mobilenet classifier demo and SSD object detection demo on the Linux platform using hardware acceleration.
3）Android Demo: Compile the SSD object detection demo on the Android platform using hardware acceleration.

2 System Dependencies Description

2.1 Linux Platform Dependencies

This Linux Arm version of API SDK is developed based on RK3399Pro 64-bit Linux, needs to be used on 64-bit Linux Arm system.

This Linux X86 version of API SDK is developed based on X86 Ubunut16.04 64-bit, needs to be used on X86 64-bit Linux system. E.g. Ubuntu16.04 64-bit X86 PC, and need make ensuring that RK1808 is connected to the PC via USB.

2.2 Android Platform Dependencies

This Android version of API SDK is developed based on RK3399Pro Android8.1, needs to be used on Android8.1 system or higher.

3 API Instructions

RKNN API is a set of application programming interfaces (APIs) that based on NPU hardware acceleration, developers can use this API to develop AI related applications, the API will call the NPU hardware accelerator.

Currently the RKNN API on the Linux and Android platforms are the same.

On the Linux platform, The API SDK provides two demos that use RKNN API, one is image classifier demo based on MobileNet model, the other is object detection demo based on SSD model.

On the Android platform, The API SDK provides one object detection demo based on SSD model.

3.1 RKNN API Details

RKNN API is a set of generic APIs designed by Rockchip for NPU hardware accelerator. This API need to be used in conjunction with RKNN-Toolkit provided by Rockchip. The RKNN-Toolkit can convert common model formats into RKNN models, such as TensorFlow models, Caffe models, etc.

A detailed description of the RKNN-Toolkit can be found in the <RKNN-Toolkit User Guide>.

The RKNN-Toolkit can generate a model file with the rknn suffix, such as *mobilenet_v1-tf.rknn*.

On the Linux platform, enter the *<rknn-api>/Linux/rknn_api_sdk* directory, the API definition is in *<rknn_api_sdk>/rknn_api/include/rknn_api.h*, and the dynamic library path of RKNN API is *<rknn_api_sdk>/rknn_api/lib(64)/librknn_api.so*. Users only need to use the header file and dynamic library in the AI application.

On the Android platform, enter the *<rknn-api>/Android/rknn_api* directory, the API definition is in *<rknn_api>/include/rknn_api*, the dynamic library path of RKNN API is *<rknn_api>/lib(64)/librknn_api.so*. Users only need to use the header file and dynamic library in the JNI library of the AI application. Currently, only JNI development methods are supported on Android.

The following section is a description of RKNN API.

3.1.1 rknn_init & rknn_init2

API	int rknn_init(rknn_context* context, void* model, uint32_t size, uint32_t flag)

	int rknn_init2(rknn_context* context, void* model, uint32_t size, uint32_t flag, rknn_init_extend* extend)
Description	Create a context and load the rknn model.
Parameter	rknn_context* context: The pointer of context object. Used to return the created context object.
	void* model: A pointer to the rknn model.
	uint32_t size: The size of the rknn model.
	uint32_t flag: Extended flag: **RKNN_FLAG_PRIOR_HIGH:** Create a high priority context. **RKNN_FLAG_PRIOR_MEDIUM:** Create a medium priority context. **RKNN_FLAG_PRIOR_LOW:** Create a low priority context. **RKNN_FLAG_ASYNC_MASK:** Enable Asynchronous mode. When enable, *rknn_outputs_get* will not block for too long, because it returns the inference result of the previous frame directly (except for the inference result of the first frame), which will significantly improve the inference frame rate in single-thread mode, but the cost is that *rknn_outputs_get* return not the inference results of the current frame. When *rknn_run* and *rknn_outputs_get* are in different threads, there is no need to enable the Asynchronous mode. **RKNN_FLAG_COLLECT_PERF_MASK:** Enable performance collection debug mode. When enable, you can query the running time of each layer of network through the *rknn_query* interface. It should be noted that the total time spent in inferring one frame is longer than *RKNN_FLAG_COLLECT_PERF_MASK* unset, because the execution of each layer needs to be synchronized.
	rknn_init_extend* extend: the pointer of extend information. Used to set or get information corresponding to the current *rknn_init*, such as *device_id* (see the rknn_init_extend definition for details). If not used, can be set to NULL.

Return	Error code (see Error Code).

The sample code is as follow:

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0);
```

3.1.2 rknn_destroy

API	int rknn_destroy(rknn_context context)
Description	Unload the rknn model and destroy the context and its associated resource.
Parameter	rknn_context context: The object of context.
Return	Error code (see Error Code).

The sample code is as follow:

```
int ret = rknn_destroy (ctx);
```

3.1.3 rknn_query

API	int rknn_query(rknn_context context, rknn_query_cmd cmd, void* info, uint32_t size)
Description	Query the related information of RKNN Model and SDK.
Parameter	rknn_context context: The object of context.
	rknn_query_cmd cmd: The command of query.
	void* info: The structure variable that store the returned result.
	uint32_t size: The size of the structure variable corresponding to ***info***.
Return	Error code (see Error Code).

The supported query commands of current SDK are shown in the following table:

Command of Query	Returned Structure	Description

RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	Query the number of input and output tensor.
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	Query the attribute of input tensor.
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	Query the attribute of output tensor.
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	Query the running time of each layer of the network. This query requires use the *RKNN_FLAG_COLLECT_PERF_MASK* in *rknn_init*, otherwise no detailed layer performance information can be obtained. In addition, the *rknn_perf_detail.perf_data* returned by the *RKNN_QUERY_PERF_DETAIL* query does not require the user to free actively. Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.
RKNN_QUERY_PERF_RUN	rknn_perf_run	Query the hardware execution time of single inference. Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	Query the SDK version.

The next section will explain in detail how each query command should be used.

3.1.3.1 Query the number of input/output tensor

The *RKNN_QUERY_IN_OUT_NUM* command can be used to query the number of input/output tensor. The object of *rknn_input_output_num* structure needs to be created first.

The sample code is as follows:

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input,
                                io_num.n_output);
```

3.1.3.2 Query the attribute of input tensor

The *RKNN_QUERY_INPUT_ATTR* command can be used to query the attribute of input tensor. The object of *rknn_tensor_attr* structure needs to be created first.

The sample code is as follows:

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
        input_attrs[i].index = i;
        ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                                        sizeof(rknn_tensor_attr));
}
```

3.1.3.3 Query the attribute of output tensor

The *RKNN_QUERY_OUTPUT_ATTR* command can be used to query the attribute of output tensor. The object of *rknn_tensor_attr* structure needs to be created first.

The sample code is as follows:

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
        output_attrs[i].index = i;
        ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                                        sizeof(rknn_tensor_attr));
}
```

3.1.3.4 Query the running time of each layer of the network

If you have set *RKNN_FLAG_COLLECT_PERF_MASK* flag when rknn_init function is called, then you can use *RKNN_QUERY_PERF_DETAIL* to query the running time of each layer of the network after the rknn_outputs_get execution completed.

The object of *rknn_perf_detail* structure needs to be created first.

In addition, the *rknn_perf_detail.perf_data* returned by the *RKNN_QUERY_PERF_DETAIL* query does not require the user to free it.

Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.

The sample code is as follows:

```
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
                                        sizeof(rknn_perf_detail));
printf("%s", perf_detail.perf_data);
```

3.1.3.5 Query the hardware execution time of single inference.

The *RKNN_QUERY_PERF_RUN* command can be used to query the hardware execution time of single inference. The object of *rknn_perf_run* structure needs to be created first.

Pay attention that the query can only return the correct query result after the *rknn_outputs_get* function is called.

The sample code is as follows:

```
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                                    sizeof(rknn_perf_run));
printf("%ld", perf_run.run_duration);
```

3.1.3.6 Query the SDK version

The *RKNN_QUERY_SDK_VERSION* command can be used to query the SDK version. The object of *rknn_sdk_version* structure needs to be created first.

The sample code is as follows:

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                                sizeof(rknn_sdk_version));
printf("api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

3.1.4 rknn_inputs_set

API	int rknn_inputs_set(rknn_context context, uint32_t n_inputs, rknn_input inputs[])
Description	Set the buffer pointer and other parameters of inputs. The buffer pointer and parameters of single input need to be stored in *rknn_input*. This function can support multiple inputs.
Parameter	rknn_context context: the object of context.
	uint32_t n_inputs: the number of inputs.
	rknn_input inputs[]: the arrays of inputs information, each element of the array is a *rknn_input* structure object.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].pass_through = FALSE;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

ret = rknn_inputs_set(ctx, 1, inputs);
```

For more detailed usage, see the step 4 of the [RKNN API Basic Call Flow] section.

3.1.5 rknn_run

API	int rknn_run(rknn_context context, rknn_run_extend* extend)
Description	Perform a model inference. The input data need to be set by *rknn_inputs_set* function before calling *rknn_run*. The *rknn_run* will not block normally, but it will block when there are more than 3 inference results not obtained by *rknn_outputs_get*.
Parameter	rknn_context context: the object of context.
	rknn_run_extend* extend: the pointer of extend information. Used to set or get information about the frame corresponding to the current *rknn_run*, such as *frame_id* (see the *rknn_run_extend* definition for details). If not used, can be set to NULL.
Return	Error code (see Error Code).

The sample code is as follows:

```
ret = rknn_run(ctx, NULL);
```

3.1.6 rknn_outputs_get

API	int rknn_outputs_get(rknn_context context, uint32_t n_outputs, rknn_output outputs[],

	rknn_output_extend* extend)
Description	Waiting for the inference operation to completed and get the output results. This function can obtain multiple output data at one time. Each output corresponds to a rknn_output structure object, you need to create and set each rknn_output object in turn before the function is called. In addition, the function will block until the inference completed (unless there is an exception error). The output results will eventually be stored in the array of *outputs[]*. There are two ways to use the buffer of the output data: 1. Users malloc and free the output buffer themselves. In this mode, the *is_prealloc* of the *rknn_output* object needs to be set to TRUE, and the *rknn_output.buf* also needs to be set by user. 2. The output buffer malloc and free by rknn api. In this mode, the *is_prealloc* of the *rknn_output* object needs to be set to FALSE, and *rknn_output.buf* will point to output data after the function is called.
Parameter	rknn_context context: the object of context.
	uint32_t n_outputs: the number of output arrays. This number must be the same as the number of outputs of rknn model. (the number of outputs of rknn model can be queried by *rknn_query*.)
	rknn_output outputs[]: the arrays of outputs information. Each element of array is a *rknn_output* structure object, representing an output of the model.
	rknn_output_extend* extend: the pointer of extend information. Used to set or get information about the frame corresponding to the current *rknn_outputs_get*, such as *frame_id* (see the *rknn_output_extend* definition for details). If not used, can be set to NULL.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
        outputs[i].want_float = TRUE;
        outputs[i].is_prealloc = FALSE;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

For more detailed usage, see the step 6 of the [RKNN API Basic Call Flow] section.

3.1.7 rknn_outputs_release

API	int rknn_outputs_release(rknn_context context, uint32_t n_ouputs, rknn_output outputs[])
Description	Release outputs that obtained by *rknn_outputs_get*. When the outputs are no longer used, you need to call the function to release it. (Whether *rknn_output[x].is_prealloc* is TRUE or FALSE, you need to call the function to release the outputs.) After the function is called: when *rknn_output[x].is_prealloc* = FALSE, the *rknn_output[x].buf* obtained by *rknn_outputs_get* is also released automatically; when *rknn_output[x].is_prealloc* = TURE, the *rknn_output[x].buf* requires user to free it.
Parameter	rknn_context context: the object of context.
	uint32_t n_outputs: the number of output arrays. This number must be the same as the number of outputs of rknn model. (the number of outputs of rknn model can be queried by *rknn_query*.)
	rknn_output outputs[]: the arrays of outputs information.
Return	Error code (see Error Code).

The sample code is as follows:

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

3.1.8 rknn_find_devices

API	int rknn_find_devices(rknn_devices_id* pdevs)
Description	find the devices information that connected to host.
Parameter	rknn_devices_id* pdevs: the pointer of devices information structure.
Return	Error code (see Error Code).

The sample code is as follows:

```
rknn_devices_id devids;
ret = rknn_find_devices (&devids);
printf("n_devices = %d\n", devids.n_devices);
for(int i=0; i<devids.n_devices; i++) {
    printf("%d:  type=%s, id=%s\n", i, devids.types[i], devids.ids[i]);
}
```

3.2 RKNN Data Structure Definition

3.2.1 rknn_input_output_num

The structure *rknn_input_output_num* represents the number of tensors of input and output. The following table shows the definition of the structure:

Field	Data Type	Meaning
n_input	uint32_t	The number of input tensor.
n_output	uint32_t	The number of output tensor.

3.2.2 rknn_tensor_attr

The structure *rknn_tensor_attr* represents the tensor attribute of rknn model, The following table shows the definition of the structure:

Field	Data Type	Meaning
index	uint32_t	The index of input or output tensor. The index needs to be set before calling the *rknn_query*.
n_dims	uint32_t	The number of tensor dimensions.
dims	uint32_t[]	Each dimension value of tensor.
name	char[]	Name of tensor.
n_elems	uint32_t	The number of tensor elements.
size	uint32_t	The memory size of tensor data.
fmt	rknn_tensor_format	The dimension format of tensor, as follows: ***RKNN_TENSOR_NCHW*** ***RKNN_TENSOR_NHWC***
type	rknn_tensor_type	The data type of tensor, as follows: ***RKNN_TENSOR_FLOAT32*** ***RKNN_TENSOR_FLOAT16*** ***RKNN_TENSOR_INT8*** ***RKNN_TENSOR_UINT8*** ***RKNN_TENSOR_INT16***
qnt_type	rknn_tensor_qnt_type	The quantization type of tensor, ds: ***RKNN_TENSOR_QNT_NONE****:* none quantization. ***RKNN_TENSOR_QNT_DFP****:* Dynamic fixed-point quantization. ***RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC****:* Asymmetric affine quantization.

fl	int8_t	Fractional length for *RKNN_TENSOR_QNT_DFP*.
zp	uint32_t	Zero point for *RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC*.
scale	float	Scale for *RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC*.

3.2.3 rknn_input

The structure *rknn_input* represents an input data of model, used as a parameter to the *rknn_inputs_set* function. The following table shows the definition of the structure:

Field	Data Type	Meaning
index	uint32_t	The index of input tensor.
buf	void*	The buffer point of input data.
size	uint32_t	The memory size of input data buffer.
pass_through	uint8_t	The pass-through mode of input. **TRUE:** The input data is passed directly to the input node of rknn model without any conversion, therefore the following ***type*** and ***fmt*** do not need to be set. **FALSE:** The input data will convert to the same data type and format as the input node of the rknn mode according to the following ***type*** and ***fmt***, therefore the following ***type*** and ***fmt*** need to be set.
type	rknn_tensor_type	The data type of input tensor, as follow: ***RKNN_TENSOR_FLOAT32*** ***RKNN_TENSOR_FLOAT16*** ***RKNN_TENSOR_INT8***

		RKNN_TENSOR_UINT8 ***RKNN_TENSOR_INT16***
fmt	rknn_tensor_format	The dimension format of input tensor, as follow: ***RKNN_TENSOR_NCHW*** ***RKNN_TENSOR_NHWC***

3.2.4 rknn_output

The structure *rknn_output* represents an output data of the model, used as a parameter to the *rknn_outputs_get* function. The following table shows the definition of the structure:

Field	Data Type	Meaning
want_float	uint8_t	Identifies whether the output data needs to be converted to float32 type.
is_prealloc	uint8_t	Identifies whether the buffer that holds the output data is pre-allocated.
index	uint32_t	The index of output tensor.
buf	void*	The buffer pointer of output.
size	uint32_t	The memory size of output data buffer.

When the *is_prealloc* is FALSE, the ***index/buf/size*** of *rknn_output* will be set after *rknn_outputs_get* is called, therefore the three members do not need to be pre-set.

When the *is_prealloc* is TRUE, the ***index/buf/size*** of *rknn_output* need to be set before calling *rknn_outputs_get*, otherwise the *rknn_outputs_get* function will fail with an error.

3.2.5 rknn_perf_detail

The structure *rknn_perf_detail* represents the performance details of rknn model. The following table shows the definition of the structure:

Field	Data Type	Meaning

perf_data	char*	Contains the running time of each layer of the network, can be printed directly for viewing.
data_len	uint64_t	The string length of ***perf_data***.

3.2.6 rknn_perf_run

The structure *rknn_perf_run* represents the execution time of a single inference of rknn model. The following table shows the definition of the structure:

Field	Data Type	Meaning
run_duration	int64_t	The hardware execution time (us) of a single inference of rknn model.

3.2.7 rknn_init_extend

The structure *rknn_init_extend* represents the extended information of rknn_init, used as parameter to rknn_init function.

The following table shows the definition of the structure:

Field	Data Type	Meaning
device_id	char*	Used to select the connected device. Such as "0123456789ABCDEF", the device id can be query by "adb devices". If only one device connected, can set nullptr.

3.2.8 rknn_run_extend

The structure *rknn_run_extend* represents the extended information of rknn_run, used as parameter to rknn_run function.

The following table shows the definition of the structure:

Field	Data Type	Meaning

frame_id	uint64_t	Used to get the frame id after the *rknn_run* function is called. The *frame_id* corresponds to *rknn_output_extend.frame_id* one by one, In the case where *rknn_run* and *rknn_outputs_get* are in different threads, it can be used to determine the correspondence of frame.

3.2.9 rknn_output_extend

The structure *rknn_output_extend* represents the extend information of *rknn_outputs_get*, used as parameter to *rknn_outputs_get* function. The following table shows the definition of the structure:

Field	Data Type	Meaning
frame_id	uint64_t	Used to get the frame id after the *rknn_outputs_get* function is called. The *frame_id* corresponds to *rknn_run_extend.frame_id* one by one, In the case where *rknn_run* and *rknn_outputs_get* are in different threads, it can be used to determine the correspondence of frame.

3.2.10 rknn_sdk_version

The structure *rknn_sdk_version* represents the version information of RKNN SDK. The following table shows the definition of the structure:

Field	Data Type	Meaning
api_version	char[]	The version of RKNN API.
drv_version	char[]	The driver version on which RKNN API is based.

3.2.11 rknn_devices_id

The structure *rknn_devices_id* represents the information of device ID list. The following table shows the definition of the structure:

Field	Data Type	Meaning
n_devices	uint32_t	The number of devices
types	char[][]	The array of device type.
ids	char[][]	The array of device ID.

3.2.12 Error Code

The return error code of RKNN API. The following table shows the definition:

Error Code	Meaning
RKNN_SUCC	Execution is successful.
RKNN_ERR_FAIL	Execution is failed.
RKNN_ERR_TIMEOUT	Execution timeout.
RKNN_ERR_DEVICE_UNAVAILABLE	The NPU device is unavailable.
RKNN_ERR_MALLOC_FAIL	Memory allocation is failed.
RKNN_ERR_PARAM_INVALID	The parameter is invalid.
RKNN_ERR_MODEL_INVALID	The RKNN model is invalid.
RKNN_ERR_CTX_INVALID	The rknn_context is invalid.
RKNN_ERR_INPUT_INVALID	The object of rknn_input is invalid.
RKNN_ERR_OUTPUT_INVALID	The object of rknn_output is invalid.
RKNN_ERR_DEVICE_UNMATCH	The device version does not match.

3.3 RKNN API Basic Call Flow

1）Load the file of rknn model into memory, the file of rknn model is a model file with the rknn

suffix generated by the RKNN-Toolkit that described above, such as *mobilenet_v1-tf.rknn*.

2）Call the *rknn_init* to initialize the context and load the rknn model, code is as follows:

```
rknn_context ctx = 0;
ret = rknn_init(&ctx, model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ret < 0) {
    printf("rknn_init fail! ret=%d\n", ret);
    goto Error;
}
```

The *ctx* is the context object; the *model* is the pointer of rknn model in memory; the *model_len* is size of model; the *RKNN_FLAG_PRIOR_MEDIUM* is the priority flag.

3）The attributes of input/output of rknn model may be different from the original model (pb or caffe), so you need to get the new attributes of input/output through *rknn_query* function, as follows:

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n",ret);
    goto Error;
}
```

The above code used to get the number of input and output, the number will store in *io_num.n_input* and *io_num.n_output*.

Next get the attribute of output:

```
    rknn_tensor_attr output0_attr;
    output0_attr.index = 0;
    ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &output0_attr,
sizeof(output0_attr));
    if(ret < 0) {
        printf("rknn_query fail! ret=%d\n",ret);
        goto Error;
    }
```

The above code used to get the attribute of an output, remember to set the index of *rknn_tensor_attr* (the index cannot be greater than or equal to the number of outputs that obtained earlier).

Obtaining an input attribute method is similar to getting the output attribute method.

4) Call *rknn_input_set* to set the inputs according to the input parameter/format of rknn model, code is as follows:

```
    rknn_input inputs[1];
    inputs[0].index = input_index;
    inputs[0].buf = img.data;
    inputs[0].size = img_width * img_height * img_channels;
    inputs[0].pass_through = FALSE;
    inputs[0].type = RKNN_TENSOR_UINT8;
    inputs[0].fmt = RKNN_TENSOR_NHWC;
    ret = rknn_inputs_set(ctx, 1, inputs);
    if(ret < 0) {
        printf("rknn_input_set fail! ret=%d\n", ret);
        goto Error;
    }
```

First create an array of *rknn_input* (here assumes that there is only one input, so the array size is set to 1), then fill each member of each array item:

inputs[0].index Index of input node.

inputs[0].buf Buffer pointer that can be accessed by cpu, generally pointer to image data that generated by camera, such as RGB888 data.

inputs[0].size The size of buffer.

inputs[0].pass_through Pass-through mode:

TRUE: If the attributes (mainly ***type***, ***fmt*** and the quantization parameter) of input data are consistent with the input attributes obtained by the rknn_query, then the ***pass_through*** can be set to TRUE, and the following ***type*** and ***fmt*** don't need to be set. In this mode, *rknn_inputs_set* will pass the input data directly to the input node of rknn model. This mode is used by user to know the input attribute of the rknn model, and has converted the original input data to the data that consistent with the rknn model input.

FALSE: If the attributes (mainly ***type***, ***fmt*** and the quantization parameter) of input data are inconsistent with the input attributes obtained by the *rknn_query*, then the ***pass_through*** needs to be set to FALSE, and the following ***type*** and ***fmt*** also need to be set by user. In this mode, the *rknn_inputs_set* function will perform type and format conversion and quantization processing automatically. Note that this mode does not support dynamic fixed point (DFP) or asymmetric affine (AFFINE ASYMMETRIC) input data passed by user.

inputs[0].type Data type of buffer, if it is RGB888 data, then set to *RKNN_TENSOR_UINT8*.

inputs[0].fmt Data format of buffer, that is NHWC or NCHW, the data format that obtained by camera is generally *RKNN_TENSOR_NHWC*.

5）Call *rknn_run* to trigger the inference operation after the input parameter was set. The function will return immediately (but when there are more than 3 inference results not obtained by *rknn_outputs_get*, the rknn_run will block until the *rknn_outputs_get* is called). Code is as follows:

```
ret = rknn_run(ctx, NULL);
if(ret < 0) {
    printf("rknn_run fail! ret=%d\n", ret);
    goto Error;
}
```

6）Now you can call *rknn_outputs_get* to wait for the inference to complete after *rknn_run* is called, the *rknn_outputs_get* will block until the inference is completed, and then the inference results can be obtained. Code is as follows:

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = FALSE;
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

First create the array of *rknn_output* (assume there is only one output, so the size of array set to 1). The first two members of *rknn_output* need to be set, namely *outputs[0].**want_float*** and *outputs[0].**is_prealloc***.

want_float**:* Since the output type of the rknn model may be inconsistent with the output type of the original model. In general, the output type of rknn model is UINT8 or FP16 (the output specific attribute of rknn model can be obtained by *rknn_query*). If the user wants to obtain the FP32 output data, the ***want_float can be set to TRUE; If the user wants to get the raw output data of rknn model, set it to FALSE.

is_prealloc* = FALSE**: If the user does not pre-allocate the buffer of each output, the ***is_prealloc flag can be set to FALSE, and the remaining member of *outputs[0]* do not need to be set. The inference results will be stored in *output[0]* after *rknn_outputs_get* returned, the

results contain:

outputs[0].index Index of output node.

outputs[0].buf Buffer pointer that store inference result.

outputs[0].size Size of buffer.

In addition, the other attribute of inference result of *output[0]* can be obtained by *rknn_query*. It should be noted that the *outputs[0].buf* is automatically released when the *rknn_output_release* is called, so there is no need to free it by user.

is_prealloc* = TRUE**: If the user has pre-allocate the buffer of each output, the ***is_prealloc flag can be set to TRUE, and the remaining member of *output[0]* also need to be set Code is as follows:

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = TRUE;
outputs[0].index = 0;
outputs[0].buf = output0_buf;
outputs[0].size = output0_attr.n_elems * sizeof(float);
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

The remaining member of *output[0]* is:

outputs[0].index Index of output node. The user needs to specify the index of the output, and the index must be smaller than the number of outputs of rknn model. (the number of outputs of rknn model can be obtained by *rknn_query*.)

outputs[0].buf Buffer pointer for store inference result. The buf need to be pre-created by user.

outputs[0].size Size of buffer. The size needs to be calculated according to the corresponding output attribute and the ***want_float*** flag.

When ***want_float*** is FASLE, the **size** equal to the *output0_attr.size*;

When ***want_float*** is FALSE, the **size** equal to:

*output0_attr.n_elems * sizeof(float)*.

(*output0_attr* is attribute of output 0 that obtained by *rknn_query*.)

After the *rknn_outputs_get* is returned, the inference result of corresponding index will be stored in the *output[0].buf*, since the buf is created by user, so the user needs to free it to avoid memory leak when it is no longer needed.

7）When all the outputs obtained by *rknn_outputs_get* are no longer needed, you need to call *rknn_outputs_release* to release the outputs, otherwise it will cause a memory leak. Code is as follows:

```
rknn_outputs_release(ctx, 1, outputs);
```

The way of passing parameter is similar to *rknn_outputs_get*.

It should be noted that whether the *rknn_output[x].is_prealloc* is TRUE or FALSE, this function needs to be called to release the output finally.

8）If you need to make multiple inferences, you can jump back to step 4 for next inference.

9）When the program needs to exit, you need to call *rknn_destroy* to unload model and destroy the context, code is as follows:

```
rknn_destroy(ctx);
```

For more detailed code, please refer to the file of API SDK under the Linux directory:

<Linux>/rknn_api_sdk/rknn_mobilenet.cpp

<Linux>/rknn_api_sdk/rknn_ssd.cpp

or under the Android directory:

<Android>/rk_ssd_demo/app/src/main/jni/ssd_image.cc

4 Demo Instructions

4.1 Linux Arm Demo

4.1.1 Compilation Instructions

Two demos using RKNN API are provided in Linux directory of API SDK, one is image classifier demo based on MobileNet, the other is object detection demo based on SSD.

Enter the *<Linux>/rknn_api_sdk* directory, the main source file for these two demos is *<rknn_api_sdk>/rknn_mobilenet.cpp* and *<rknn_api_sdk>/rknn_ssd.cpp*, the specific compile method is as follows:

1）Download the arm cross-complication tool and configure it:
https://releases.linaro.org/components/toolchain/binaries/6.3-2017.05/aarch64-linux-gnu/

2）cd rknn_api_sdk; mkdir build_arm; cd build_arm

3）cmake -DCMAKE_SYSTEM_NAME=Linux -DCMAKE_C_COMPILER=aarch64-linux-gnu-gcc -DCMAKE_CXX_COMPILER=aarch64-linux-gnu-g++

4）make

You can get *rknn_mobilenet* and *rknn_ssd* executable file in *<rknn_api_sdk>/build_arm/* after the make is finished.

Note: Currently the demo is only available for the Linux Arm 64-bit system, so only 64-bit rknn api library is provided. The demo is verified on the RK3399Pro Linux 64-bit system.

4.1.2 Run Instructions

For running the *rknn_mobilenet* and *rknn_ssd*, you need to copy the dependencies library to *<Target*

Root>/usr/lib/ or <*Target Root*>/usr/lib64/, and copy the relevant resource files to the <*Target Root*>*/tmp* directory. The specific steps are as follows:

1）Copy the contents in the <*Linux*>*/rknn_api_sdk/3rdparty/opencv/arm/lib64* directory and <*Linux*>*/rknn_api_sdk/rknn_api/arm/lib64* directory to the */usr/lib/ or /usr/lib64/* directory on the target board.

2）Copy the contents in the <*Linux*>*/tmp/* directory of the API SDK package to the */tmp/* directory of the target board.

3）Copy the *rknn_mobilenet* and *rknn_ssd* compiled in <*Linux*>*/rknn_api_sdk/build_arm* directory to the /tmp/ directory of the target board.

4）Go to the /tmp directory of the target board to execute:

./rknn_mobilenet

After the execution is successful, it will print the execution time and results.

Go to the */tmp* directory of the target board to execute:

./rknn_ssd

After the execution is successful, it will print the execution time and results. At the same time, the image *out.jpg* containing the detection result will be generated in the */tmp* directory of the target board, you can export the *out.jpg* to view the detection result.

4.2 Linux X86 Demo

4.2.1 Compilation Instructions

Two demos using RKNN API are provided in Linux directory of API SDK, one is image classifier demo based on MobileNet, the other is object detection demo based on SSD.

Enter the <*Linux*>*/rknn_api_sdk* directory, the main source file for these two demos is <*rknn_api_sdk*>*/rknn_mobilenet.cpp* and <*rknn_api_sdk*>*/rknn_ssd.cpp*, the specific compile

method is as follows:

1）cd rknn_api_sdk; mkdir build_x86; cd build_x86; cmake ..

2）make

You can get *rknn_mobilenet* and *rknn_ssd* executable file in *<rknn_api_sdk>/build_x86/* after the make is finished.

Note: Currently the demo is only available for the X86 Linux 64-bit system, so only 64-bit rknn api library is provided. The demo is verified on the Ubuntu 16.04 64-bit system.

4.2.2 Run Instructions

For running the *rknn_mobilenet* and *rknn_ssd*, you need to copy the dependencies library and the relevant resource files to the /tmp directory. The specific steps are as follows:

1）Copy the contents in the *<Linux>/rknn_api_sdk/3rdparty/opencv/x86/lib64* directory and *<Linux>/rknn_api_sdk/rknn_api/x86/lib64* directory to the /tmp/ directory on the x86 system.

2）Copy the contents in the *<Linux>/tmp/* directory of the API SDK package to the */tmp/* directory on the x86 system.

3）Copy the *rknn_mobilenet* and *rknn_ssd* compiled in *<Linux>/rknn_api_sdk/build_x86* directory to the */tmp/* directory on the x86 system.

4）Copy the *npu_transfer_proxy* in *<npu_transfer_proxy>/linux-x86_64* directory to the */tmp/* directory on the x86 system.

5）Make sure that the RK1808 is connected to the PC via USB, and you can see the following device information through ‘lsusb’:

```
Bus 001 Device 032: ID 2207:0019
```

6）Go to the /tmp directory to execute:

sudo ./npu_transfer_proxy &

```
export LD_LIBRARY_PATH=/tmp
./rknn_mobilenet
```

After the execution is successful, it will print the execution time and results.

```
export LD_LIBRARY_PATH=/tmp
./rknn_ssd
```

After the execution is successful, it will print the execution time and results. At the same time, the image *out.jpg* containing the detection result will be generated in the */tmp* directory, you can open the *out.jpg* to view the detection result.

4.3 Android Demo

4.3.1 Compilation Instructions

There are *<Android>/rknn_api* directory and *<Android>/rk_ssd_demo* directory under the Android directory of API SDK.

rknn_api directory:

If you want to use RKNN API directly to develop your own JNI library, the JNI library can include the *<Android>/rknn_api/include/rknn_api.h* and *<Android>/rknn_api/lib(64)/librknn_api.so* to call RKNN API.

rk_ssd_demo directory:

The directory is an object detection demo based on the SSD using RKNN API. The demo contains the java and JNI parts. The JNI directory is *<Android>/rk_ssd_demo/app/src/main/jni*, the *rknn_api.h* header file and the *librknn_api.so* library file are already included in the JNI directory.

The specific compilation method of *rk_ssd_demo* is as follows:

1) Enter the <*Android*>/*rk_ssd_demo* directory, and open the project file by Android Studio.

2) Build and generate apk (need NDK support, verified on *android-ndk-r16b*).

4.3.2 Run Instructions

Run the apk directly on Android. (The demo needs an onboard camera or an external USB camera support.)

5 Appendix

5.1 API Migration Instructions

Since the API changes made from v0.9.1 to v0.9.2 are relatively large, users can migrate the codes according to the following migration steps and the above API description. The general steps are as follows:

1）Since the definition of the context handle is changed from *int* type to the *rknn_context* type, so the context variable and the use of *rknn_init* are slightly changed. The codes can be modified from:

```
int ret = 0;
int ctx = rknn_init(model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ctx < 0) {
    printf("rknn_init fail! ret=%d\n", ctx);
    goto Error;
}

…

if(ctx >= 0)            rknn_destroy(ctx);
```

To:

```
int ret = 0;
rknn_context ctx = 0;
ret = rknn_init(&ctx, model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ret < 0) {
    printf("rknn_init fail! ret=%d\n", ret);
    goto Error;
}

…

if(ctx)                 rknn_destroy(ctx);
```

Note: Parts in red are changed.

2）Since the *rknn_input_set* function needs to support data types and formats other than *INT8*, so

the definition of function has also been adjusted. The codes can be modified from:

```
    ret = rknn_input_set(ctx, input_index, img.data, img_width * img_height * img_channels,
RKNN_INPUT_ORDER_012);
    if(ret < 0) {
        printf("rknn_input_set fail! ret=%d\n", ret);
        goto Error;
    }
```

To:

```
    rknn_input inputs[1];
    inputs[0].index = input_index;
    inputs[0].buf = img.data;
    inputs[0].size = img_width * img_height * img_channels;
    inputs[0].pass_through = false;
    inputs[0].type = RKNN_TENSOR_UINT8;
    inputs[0].fmt = RKNN_TENSOR_NHWC;
    ret = rknn_inputs_set(ctx, 1, inputs);
    if(ret < 0) {
        printf("rknn_input_set fail! ret=%d\n", ret);
        goto Error;
    }
```

Note: Parts in red are changed. In addition, the parameter *RKNN_INPUT_ORDER_012* does not need to be used, and the *rknn_inputs_set* also has an additional *s*.

3) The *rknn_outputs_get* and *rknn_output_to_float* function are merged in v0.9.2, and added a new way to use the memory, so the change is great. The codes can be modified from:

```
    int h_output = -1;
    struct rknn_output outputs[2];
    h_output = rknn_outputs_get(ctx, 2, outputs, nullptr);
    if(h_output < 0) {
        printf("rknn_outputs_get fail! ret=%d\n", ret);
        goto Error;
    }
```

To:

```
rknn_output outputs[2];
outputs[0].want_float = true;
outputs[0].is_prealloc = false;
outputs[1].want_float = true;
outputs[1].is_prealloc = false;
ret = rknn_outputs_get(ctx, 2, outputs, nullptr);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

Note: Parts in red are changed. The number of outputs above is 2 for example, and other models can be modified based on actual conditions.

4) Since the *rknn_outputs_get* has merged the functions of *rknn_output_to_float* (that is the *want_float* flag above), so the call step of *rknn_output_to_float* can be removed. The codes can be modified from:

```
float *predictions = (float*)(outputs[0].buf);
if(outputs_attr[0].type != RKNN_TENSOR_FLOAT32) {
    predictions = (float*)malloc(output_size1);
    rknn_output_to_float(ctx, outputs[0], (void*)predictions, output_size1);
}
float *outputClasses = (float*)(outputs[1].buf);
if(outputs_attr[1].type != RKNN_TENSOR_FLOAT32) {
    outputClasses = (float*)malloc(output_size2);
    rknn_output_to_float(ctx, outputs[1], (void*)outputClasses, output_size2);
}

…

if(outputs_attr[0].type != RKNN_TENSOR_FLOAT32) {
    free(predictions);
}
if(outputs_attr[1].type != RKNN_TENSOR_FLOAT32) {
    free(outputClasses);
}
```

To:

```
float *predictions = (float*)(outputs[0].buf);
float *outputClasses = (float*)(outputs[1].buf);
```

Note: Parts in red are changed. The above is based on the post-processing of SSD, and other models can be modified based on actual conditions.

5) Because the above *want_float* is set to True when *rknn_outputs_get is* called, so the value of *outputs[x].size* may be inconsistent with the value of *outputs_attr[x].size* that queried by *rknn_query*, therefore the judgement condition that judges whether *output[x].size* is consistent with the attribute of query needs to be modified. The codes can be modified from:

```
// Process output
if(outputs[0].size == outputs_attr[0].size && outputs[1].size == outputs_attr[1].size)
{
    …
}
```

To:

```
// Process output
if(outputs[0].size == outputs_attr[0].n_elems*sizeof(float) && outputs[1].size == outputs_attr[1].n_elems*sizeof(float))
{
    …
}
```

Note: Parts in red are changed. The number of outputs above is 2 for example, and other models can be modified based on actual conditions.

6) The use of *rknn_outputs_release* has also been adjusted, the way of parameter are passed consistent with the *rknn_outputs_get* (the *h_output* does not need to use). The codes can be modified from:

```
rknn_outputs_release(ctx, h_output);
```

To:

```
rknn_outputs_release(ctx, 2, outputs);
```

Note: Parts in red are changed.